Rekurzivni upiti u PostgreSQL-u

Garić, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Garić

REKURZIVNI UPITI U POSTGRESQL-U

ZAVRŠNI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Garić

Matični broj: 44116/15–R

Studij: Informacijski sustavi

REKURZIVNI UPITI U POSTGRESQL-U

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Markus Schatten

Varaždin, rujan 2018.

Luka Garić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Današnji sustavi za upravljanje bazom podataka nemaju elegantno rješenje kod ispisivanja podataka po određenim razinama. Iz tog razloga ispisivanje hijerarhijski strukturiranih podataka zahtjeva korištenje rekurzivnih upita kako bi se efikasno ispisali svi potrebni podaci. U ovom završnom radu obradit ćemo postupak dohvaćanja takvih podataka putem rekurzivnih upita u PostgreSQL sustavu za upravljanje bazom podataka. Za pisanje takvih uvjeta potrebno je korištenje WITH RECURSIVE klauzula. Na primjerima ćemo vidjeti implementaciju rekurzivnih upita nad jednom tablicom i nad dvije tablice nakon čega ćemo usporediti razliku između rekurzivnog i iterativnog upita. Nakon usporedbe ćemo ukratko objasniti pojam beskonačne rekurzije. Rezultate svih dobivenih istraživanja žemo usporediti te zaključiti jesu li rekurzivni upiti dobro rješenje za ispis hijerarhijski strukturiranih podataka.

Ključne riječi: PostgreSQL; SQL upit; rekurzija; Common Table Expression (CTE); hijerarhijska struktura; baza podataka; WITH klauzula;

Sadržaj

1. Uvod

Izrada aplikacije, programa, web stranice ili igrice je danas nezamisliva bez baze podataka u kojoj su smješteni svi potrebni podaci za njihovo funkcioniranje. Zbog lakše komunikacije s bazom podataka stvoren je SQL proceduralni jezik. Nakon izlaska prve komercijalne verzije SQL jezika počinju se profesionalno razvijati sustavi za upravljanje bazama podataka poput PostgreSQL-a koji se koristi u ovom radu. Korištenje rekurzivnih algoritama za dohvaćanje podataka u sustavima za upravljanje bazama podataka je relativno nov koncept implementiran u SQL jezik 1999. godine. Rekurzivni algoritmi se koriste kada je potrebno ponavljanje određenog dijela algoritma kako bi se došlo do rješenja. U bazama podataka je potreba za korištenje rekurzije rijetka no postoje problemi kod dohvaćanja podataka u kojima korištenje rekurzije pojednostavljuje pisanje upita. Jedan od problema je potreba za ispisivanje podataka po određenim razinama. Dakle rekurzija je vrlo korisna ukoliko se u bazi podataka nalaze hijerarhijski strukturirani podaci poput organizacijske strukture.

Rekurzivni upiti se sastoje od minimalno dva podupita, početni upit te rekurzivni upit koji se ponavlja dok se ne dođe do rezultata. Kako bi lakše razumjeli pojam rekurzivnih upita nakon teorijskog dijela ćemo na primjeru pokazati dohvaćanje podataka iz jedne tablice te iz dvije tablice pomoću rekurzivnih upita. Svi primjeri će biti detaljno objašnjeni za dodatno razumijevanje. Svi rekurzivni upiti se također mogu napisati i na iterativni način ručnim pisanjem upita za dohvaćanje svih potrebnih podataka. Oba načina imaju svoje prednosti i nedostatke ovisno o veličini skupa podataka te broju iteracija koje se moraju izvršiti kako bi se došlo do željenog rezultata. Kako bi ispitali uspješnost rekurzivnog upita u usporedbi s upitom pisanim na iterativni način ispitat ćemo brzine oba upita nad skupovima podataka s različitim brojem razina te sa skupovima podataka različitih veličina. Rezultate istraživanja ćemo usporediti te zaključiti u kojim okolnostima se preporuča koristiti rekurzivne upite. Također ćemo prikazati uspješnost rekurzije u postotcima u odnosu na iterativni način.

Kako bi pojam rekurzivnog upita u PostgreSQL-u bio u potpunosti jasan prvo ćemo proći kroz sami pojam rekurzije kao algoritma za rješavanje problema. Nakon toga ćemo se upoznati s proceduralnim jezikom zaduženim za komunikaciju s bazom podataka te sustavom za upravljanje bazom podataka. Također ćemo objasniti korisničko sučelje koje je korišteno tijekom pisanja rada. Kada se upoznamo sa svim potrebnim pojmovima objasnit ćemo što su to rekurzivni upiti te kako ih implementirati u PostgreSQL-u. Nakon primjera slijedi usporedba brzine iterativnih i rekurzivnih upita te analiza dobivenih rezultata. Na kraju ćemo također na primjeru prikazati mogući scenarij beskonačne rekurzije te obrazložiti sve dobivene rezultate u zaključku rada.

2. Rekurzija

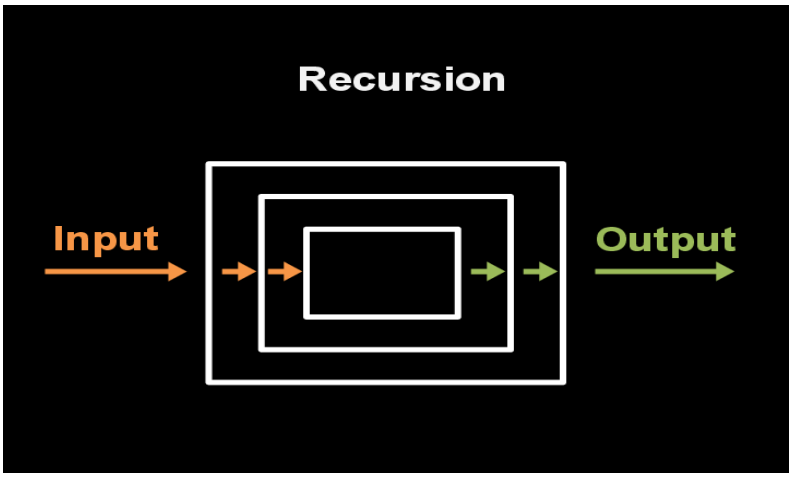
Većina problema s kojima se susrećemo se može riješiti odgovarajućim algoritmom. No postoje neki problemi koji su relativno kompleksni i opširni. Takvi problemi zahtjevaju kompleksnije algoritme koji koriste ponavljanja kako bi riješili zadani problem. Shodno tomu kompleksniji problemi zahtjevaju kompleksnije algoritme za njihovo rješavanje. Recimo da je potrebno zbrojiti sve tekstualne datoteke unutar jedne mape, takav problem je relativno jednostavan ukoliko unutar odabrane mape ne postoji niti jedna podmapa. Što se više podmapa nalazi to je problem kompleksniji za riješiti ukoliko želimo točan rezultat. Za rješavanje takvog problema potrebno je iterirati kroz svaku podmapu te zbrajati sve tekstualne datoteke dok se ne dobije konačan rezultat. Upravo takvi problemi se vrlo jednostavno rješavaju rekurzijom jer se isti algoritam ponavlja u svakoj podmapi (zbroji tekstualne datoteke). Dakle ukoliko ovaj problem želimo riješiti, najefikasnije rješenje je korištenje rekurzije.

Rekurzija se može svesti na tri glavna dijela:

- Početni korak
- Rekurzivni algoritam
- Uvjet završetka

Većina rekurzija mora sadržavati početni korak na temelju kojeg se izvršava rekurzivni algoritam. Početni korak je dakle postavljanje temelja, tj. dohvaćanje potrebnih podataka od kojih rekurzivni algoritam kreće. Da bi algoritam bio rekurzivan, mora sadržavati dio kojim poziva samog sebe. Glavni dio rekurzivnog algoritma je izvršavanje procedure koja postavlja glavno pitanje (npr. Koliko tekstualnih datoteka sadrži mapa). Po završetku procedure algoritam provjerava uvjet završetka. Ukoliko uvjet završetka nije zadovoljen obavlja se unaprijed određena procedura (npr. Nepregledana podmapa postaje glavna mapa) nakon čega algoritam poziva sam sebe te ponovno izvršava proceduru. [1]

Kada dođe do trenutka gdje nema podmapa dobiveni rezultat se vraća unatrag te na taj način ispituje sve podmape i zbraja dobivene rezultate. Nakon što su sve podmape ispitane algoritam se vratio do glavne mape s rezultatom. U ovom slučaju uvjet završetka je bilo pitanje „Postoji li podmapa koja nije ispitana?“. Dakle rekurzija je proces rješavanja problema na način da se problem razdvaja na manje verzije istog problema sve do najmanjeg problema čije rješenje je trivijalno.



Slika 1. Rekurzija (Izvor: CS50 Study, 2018)

Prema tome rekurzija je način rješavanja problema u kojem se problem razdvaja na manje instance istog problema dok se ne dođe do najmanje instance. U programskoj industriji rekurzivni algoritmi su funkcije koje pozivaju samu sebe kako bi riješili određeni problem. Takvi algoritmi ne koriste iteracije nego se oslanjaju na rekurziju. Svaki rekurzivni algoritam se može riješiti i na iterativni način, no koji algoritam se koristi ovisi o vrsti problema koji se rješava. Hijerarhijski strukturirani podaci se najčešće dohvaćaju rekurzivnim načinom zbog jednostavnosti i skalabilnosti algoritma.

Ukoliko se problem zbrajanja .txt datoteka u mapi želi riješiti rekurzijom, početni korak bi bio izračun broja datoteka u početnoj mapi. Na kraju izračuna se mora nalaziti provjera postoje li podmape, ukoliko postoji izračunaj broj datoteka u podmapi (rekurzivni algoritam). U ovom slučaju uvjet završetka je pitanje postoji li podmapa koja nije ispitana, ukoliko ne postoji rekurzija se završava te se rješenje vraća do početne mape. Na ovaj način se uz pomoć rekurzivnog algoritma može riješiti problem izračuna broja .txt datoteka bez obzira koliko podmapa postoji bez mijenjanja algoritma.

3. Structured Query Language

Prvu verziju Structured Query Language-a (kratica SQL) su razvili Donald D. Chamberlin i Raymond F. Boyce u ranih 1970-ih godina pod nazivom Structured English Query Language (kratica SEQUEL). Kasnije je SEQUEL preimenovan u SQL jer je „SEQUEL“ tada bio zaštitni znak britanske aviokompanije. Programski jezik SQL je dizajniran da olakša manipulaciju i dohvaćanje podataka pohranjenih u IBM relacijskoj bazi podataka. 1979. godine Relation Software predstavlja prvu komercijalno dostupnu SQL verziju pod nazivom Oracle V2. [Prema D. D. Chamberlin 2012]

SQL jezik je podijeljen na više elemenata koji se kombiniraju kako bi se kreirao upit za dohvaćanje podataka. Neki od elemenata su:

- Klauzule – unaprijed definirane riječi koje obavljaju određenu radnju nad bazom podataka koristeći dane parametre, izraze ili predikate (npr. WHERE)
- Izrazi – kombinacija jedne ili više vrijednosti, operatora ili funkcija koje predstavljaju određenu vrijednost (npr. „Populacija + 1“)
- Predikati – određuju uvjete koji se obrađuju (npr. true/false)
- Upiti – skup klauzula izraza i predikata u svrhu dohvaćanja, brisanja ili ažuriranja podataka. Jedan od najvažnijih elemenata SQL-a

```
UPDATE klauzula |UPDATE zavrsni.zaposlenici
SET    klauzula |SET ime = 'John'
WHERE  klauzula |WHERE id = 134;
```

Slika 2. SQL sintaksa (autorski rad)

Ukoliko je potrebno promijeniti određeni podatak u bazi podataka koristi se UPDATE klauzula. Slika 2. je primjer ažuriranja jedne instance u bazi podataka u kojoj mijenjamo ime zaposlenika u „John“ gdje je id zaposlenika jednak broju 134. U ovom slučaju „ime = 'John'“ i „id = 134“ su izrazi pomoću kojih određujemo kojem zaposleniku mijenjamo ime i koje je to ime. U ovom primjeru upit sadržava tri klauzule i dva izraza.

4. PostgreSQL

Istraživački projekt u „University of California, Berkeley“ zvan Ingres (*Interactive Graphics Retrieval System*) je započeo 1970-ih godina. Ingres je namijenjen za velike sustave te vladine aplikacije, no zbog problema sa suvremenim sustavom voditelj tima Michael Stonebraker pokreće novu verziju pod imenom „post-Ingres“ koja se kasnije preimenovala u „Postgres“ te nakon što su uveli podršku za SQL jezik postaje „PostgreSQL“. PostgreSQL zajednica je 2007. godine predložila izmjenu imena u „Postgres“ koji je i danas široko prihvaćen naziv, no PostgreSQL Core Team je odlučio zadržati postojeći naziv. [5]

PostgreSQL je objektno relacijski sustav za upravljanje bazom podataka (eng. *ORDBMS, Object Relational Database Management System)* koji se bazira na proširivosti i usklađenosti s trenutnim normama. Može se koristiti kako u malim aplikacijama na samo jednom računalu tako i u velikim aplikacijama koje koristi veliki broj korisnika. Dostupan je za Microsoft Windows i Linux dok je na MacOS uređajima zadana baza podataka. Također je u skladu s ACID pravilima koja garantiraju ispravnost transakcija čak i u nepredviđenim slučajevima.

Slika 3. PostgreSQL logo (Izvor: T. Baer, 2018)

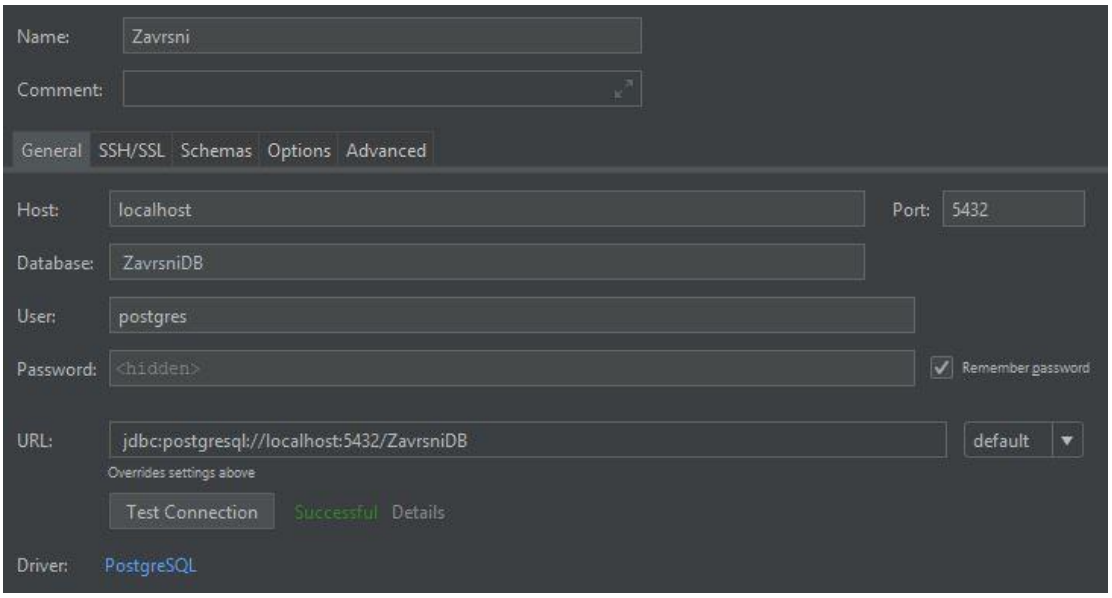
Danas je PostgreSQL jedan od najraširenijih programa za upravljanje bazom podataka zbog raznolikosti i skalabilnosti. Istraživanja obavljena 2015. godine pokazala su da je PostgreSQL (verzija 9.0) višestruko brži od MySQL-a (verzija 5.6.15) kod izdvajanja istih genomskih područja koristeći skupove od 80 000 podataka nasumičnog ljudskog DNK. (Prema: Matloob Khushi 2015)

Za korištenje PostgreSQL-a preporučeno je korisničko sučelje pgAdmin napravljeno isključivo za PostgreSQL. Korisničko sučelje se može pokrenuti na više jezika te podržava Linux, Unix, MacOS X i Windows operacijske sustave.

5. DataGrip

DataGrip je razvojno okruženje (eng. *Intergrated Development Environment,* kratica: IDE) namjenjeno za profesionalne SQL developere. Razvijeno je od strane studija JetBrains (bivši IntelliJ). Razvojno okruženje podržava većinu SQL sustava uključujući Derby, DB2, H2, HSQLDB, MySQL, Oracle, SQL Server, PostgreSQL, SQLite itd. Program sadrži opciju specificiranja SQL jezika koji će se koristiti kroz cijeli program te SQL jezika koji se koristi nad trenutnom bazom podataka.

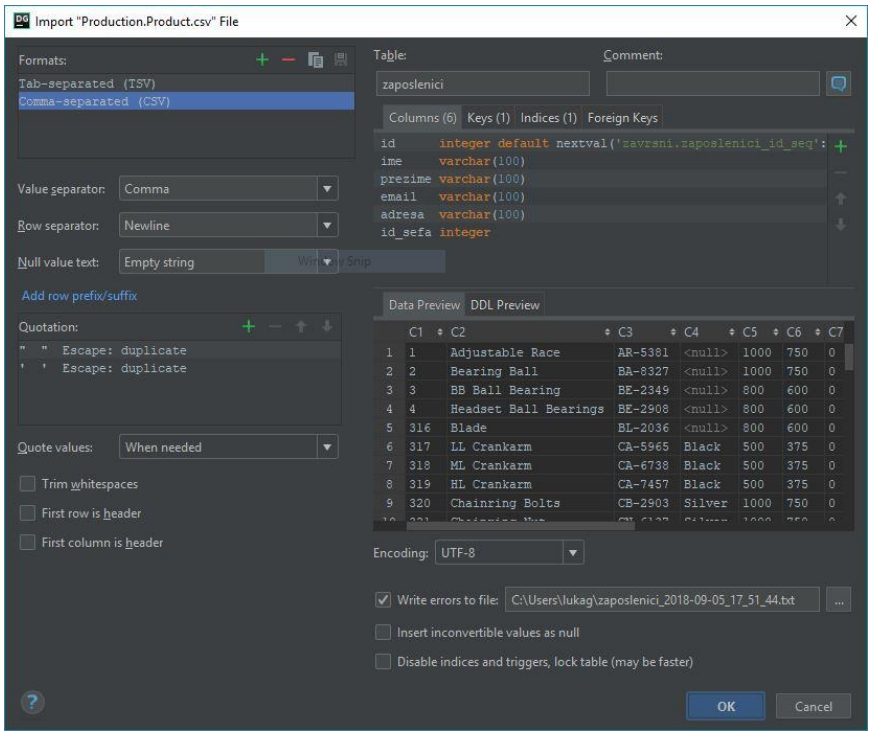
Za spajanje DataGrip alata s lokalnom PostgreSQL bazom podataka koristi se opcija „New Data Source“. Opcija izbacuje sve podržane SQL jezike gdje se odabere PostgreSQL. Nakon odabira se otvara dijalog u kojem se zapisuju podaci o serveru koji je u našem slučaju lokalni PostgreSQL server. Dakle spajamo se na localhost na default PostgreSQL port 5432. Ime baze podataka je postgres jer u ovom slučaju želimo potpunu kontrolu nad serverom.



Slika 4. Postavke za povezivanje na lokalnu PostgreSQL bazu podataka (autorski rad)

Kod velikog broja INSERT naredbi DataGrip omogućuje tzv. „batch mode“ koji izvršava 1000 upita odjednom. No takav način rada je omogućen u „Source Editor“ modu koji se otvara za svaku tablicu posebno. Desnim klikom na tablicu pojavljuje se izbornik s opcijama među kojima je opcija „SQL Scripts“ koja sadrži „Source Editor“. Unutar otvorenog prozora Source Editor-a se može zalijepiti velik broj SQL upita koji će se izvršavati u „batch“ modu. Na ovaj način su uvezeni podaci za tablicu zaposlenici jer su podaci generirani pomoću online generatora što je rezultiralo sa 137 257 INSERT naredbi.

Ugrađena opcija uvoza podataka iz datoteke je jedna od glavnih razloga za izbor DataGrip-a kao glavnog razvojnog okruženja u ovom radu. Moguće je uvesti podatke iz raznih vrsta datoteka kao što su .csv, .txt i drugi. Nakon odabira datoteke otvara se dijalog u kojem se mogu ručno izmijeniti separatori u datoteci kao i oznaka za navodnike te tekst koji zamjenjuje NULL vrijednost. DataGrip također nudi opciju označivanja prvog retka ili stupca kao zaglavlje tablice te opciju brisanja razmaka između riječi. Sve pogreške prilikom uvoza podataka se zapisuju u unaprijed definiranu tekstualnu datoteku kojoj se ručno može izmijeniti putanja i/ili naziv. Uz navedene opcije također je prikazan pregled podataka u obliku tablice uz navedene opcije mapiranja stupca iz datoteke u tablicu. Na ovaj način su uvezeni podaci iz AdventureWorks baze podataka.



Slika 5. Dijalog za uvoz podataka iz datoteke (autorski rad)

Razvijenost i jednostavnost alata je razlog zbog kojeg se njegovo korištenje naplaćuje na mjesečnoj bazi. No JetBrains svim studentima omogućuje besplatno korištenje DataGrip alata u periodu od godinu dana. Potrebno je prijaviti se s podacima od fakulteta nakon čega se dodjeljuje licenca za korištenje. Prilikom svakog pokretanja softvera ispisuje se ime i prezime te datum do kada traje dodjeljena studentska licenca. Dakle nakon isteka licence potrebno ju je obnoviti ukoliko osoba još uvijek ima status studenta.

6. Upiti u PostgreSQL-u

Svako kreiranje, dohvaćanje, ažuriranje i brisanje podataka u bazi podataka zahtjeva upotrebu određenog upita koji će izvršiti zadanu naredbu. Svi obrađivani upiti u ovom radu se odnose na PostgreSQL sintaksu pisanja upita. Većina upita nad tablicama započinje s klauzulom koja određuje koju operaciju upit izvršava. CREATE označava kreiranje novog objekta u bazi podataka (tablicu, korisnika..), SELECT označava dohvaćanje podataka iz tablice, dok ALTER i DELETE naredbe služe za ažuriranje i brisanje podataka respektivno.

6.1. Privremena tablica

Privremene tablice se kreiraju s CREATE [GLOBAL/LOCAL] TEMPORARY TABLE te im je glavna svrha privremena pohrana podataka. Postoje dvije vrste privremenih tablica: globalne i lokalne. Globalne privremene tablice su dostupne svim sesijama i poveznicama na bazu podataka te se brišu kada su sve poveznice s bazom podataka zatvorene (trenutak kada nitko nije povezan na bazu podataka). S druge strane, lokalne privremene tablice su dostupne samo jednoj sesiji i poveznici te ju vidi samo jedna osoba. Takve privremene tablice se automatski brišu na kraju sesije u kojoj su stvorene.

```
CREATE LOCAL TEMPORARY TABLE zaposlenici(
  id serial primary key,
  ime varchar(100),
  prezime varchar(100)
);

SELECT zaposlenici.ime, zaposlenici.prezime
INTO zaposlenici
FROM zavrsni.zaposlenici;
```

PostgreSQL dozvoljava kreiranje privremene tablice s imenom postojeće trajne tablice, no nije preporučeno jer tada može doći do poteškoća kod pristupanja istoimenoj trajnoj tablici. Razlog uvođenja privremenih tablica je smanjenje opterećenja baze podataka. Naime ukoliko se kreira pogled kada je potrebno privremeno doći do određenih podataka potrebno je isti obrisati nakon upotrebe. Uprotivnom se s vremenom u bazi podataka akumuliraju nepotrebni pogledi. Iz tog razloga se koriste privremene tablice kako bi se iste automatski uklanjale iz baze podataka nakon što više nisu potrebne.

6.2. WITH klauzula

WITH klauzula omogućuje kreiranje „Common Table Expression“ izraza (kratica CTE). CTE služi kao privremeni spremnik potrebnih podataka. Za razliku od privremene tablice CTE se može upotrijebiti u samo jednom izrazu koji slijedi odmah nakon upita u kojem je stvoren. CTE je dakle privremeni skup podataka nad kojim se obavlja određena radnja nakon čega se uklanja iz memorije. Primjer prikazuje jednostavnu implementaciju WITH klauzule:

```
WITH CTE_tablica (id, ime, prezime) AS (

  SELECT z.id, z.ime, z.prezime
  FROM zavrsni.zaposlenici z)

SELECT * FROM CTE_tablica;
```

Uz pomoć WITH klauzule stvara se CTE tablica nad kojom se mogu izvršiti određene operacije. U navedenom primjeru stvaramo CTE zvan CTE_tablica sa stupcnima id, ime i prezime. Nakon definiranog naziva i stupaca koje će CTE sadržavati slijedi ključna riječ AS iza koje slijedi upit naveden u zagradi. Upit mora obuhvaćati jednak broj stupaca koliko sadrži CTE tablica kako bi se podaci pravilno mapirali po stupcima. Zagrada označava kraj WITH klauzule nakon čega slijedi upit koji unutar kojeg se može referencirati na CTE tablicu. No stvoreni CTE vrijedi samo za jedan upit nakon WITH klauzule nakon čega nestaje iz memorije. Ukoliko se ne referencira nestaje iz memorije iako nije iskorišten. [8]

	id	ime	prezime
1	1	Cudmore	Emerson
2	2	Letford	Liva
3	3	Fanthom	Othelia
4	4	Benting	Hoyt
5	5	Pyvis	Izzy
6	6	Fearon	Irma
7	7	Mithun	Rozalie
8	8	Tookey	Loy
9	9	Johnathan	Cornuau
10	10	Denny	Giuroni
11	11	Rafa	Abelson
12	12	Demetrius	Ashburne
13	13	Lefty	Kupisz
14	14	Mano	Casillas

Slika 6. Rezultat WITH klauzule (autorski rad)

Dakle unutar zagrade se definira upit koji čini CTE tablicu. Nakon zagrada završava WITH klauzula i slijedi upit koji koristi CTE tablicu. MySQL omogućuje sve operacije nad CTE tablicom. Takav pristup omogućuje ažuriranje CTE tablice što rezultira ažuriranjem podataka u trajnim tablicama iz koje su se podaci mapirali u CTE. No u nekim slučajevima rezultat takvog upita može izmijeniti podatke na neželjeni način. S druge strane PostgreSQL dozvoljava upotrebu CTE tablice s isključivo SELECT klauzulom. Na taj način se izbjegavaju moguće pogreške u upitu te se sprječava moguć gubitak važnih podataka. Naprednim programerima takav pristup šteti jer im uskraćuje opcije ažuriranja podataka uz pomoć CTE tablice. No iako je korištenost CTE tablice ograničena sa samo SELECT klauzulom mogu se koristiti sve operacije (SELECT, INSERT, DELETE i UPDATE) u upitu koji slijedi odmah nakon WITH klauzule dok god je CTE upotrebljen isključivo sa SELECT klauzulom. S obzirom na to pravilo sljedeći upit se može iskoristiti:

```
INSERT INTO zavrsni.student
  SELECT 100, ime, prezime, 'M', 1
  FROM CTE_tablica
  WHERE CTE_tablica.id=1;
```

Dakle ako se poslije WITH klauzule upiše gore navedeni upit u tablicu student će se unijeti ime i prezime zaposlenika koji ima id jednak broju 1. No potrebni su još neki podaci koje tablica student zahtjeva zbog čega ih se mora ručno unijeti. Broj 100 predstavlja id koji će novi redak dobiti u tablici student te 'M' predstavlja spol osobe. Zadnji stupac u tablici student predstavlja vanjski ključ na tablicu fakultet. U ovom slučaju taj broj je ručno unesen. Nakon provedenog upita tablica student ima novi redak te izgleda kao na slici 5.

	student_id ▲ 1	ime	prezime	spol	fakultet
1	1	Petar	Perić	M	1
2	2	Ivan	Ivić	M	4
3	3	Nikolina	Jukić	M	2
4	4	Marija	Andrić	M	3
5	5	Dominik	Vukojević	M	1
6	6	Bruno	Kajmić	M	1
7	7	Mario	Miličić	M	2
8	8	Domenika	Benak	M	4
9	9	Tena	Kruljac	M	3
10	100	Cudmore	Emerson	M	1

Slika 7. Rezultat INSERT klauzule s CTE tablicom

7. Algoritmi pretraživanja binarnog stabla

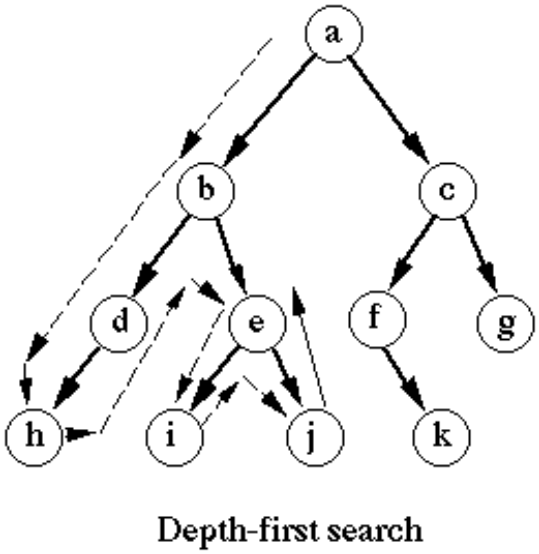
Binarno stablo je vrsta hijerarhijske strukture koja se sastoji čvorova. Čvor na najvišoj razini se naziva korjen stabla. Svaki čvor stabla se može granati na više čvorova pomoću tzv. grana koji čine podskup stabla (podstablo). Krajnji čvorovi koji se nalaze na najnižoj razini se nazivaju listovi. Dakle stablo se može razdvojiti na više podstabala koji se mogu smatrati kao zasebna stabla. Po tome je sama definicija stabla rekurzivna. Ovakav način zapisivanja podataka je prikladan za zapisivanje hijerarhijski strukturiranih podataka.

Prema V. S. Adamchik najveće prednosti binarnog stabla su mogućnost pohrane strukturirano povezanih podataka i hijerarhijski strukturiranih podataka. Također omogućuju efikasno umetanje i pretraživanje podataka. Kao jedno od najvažnijih prednosti navodi fleksibilnost podataka koja omogućuje vrlo jednostavno premještanje podstabala od jednog do drugog mjesta. Upravo zbog tih prednosti je binarno stablo jedno od najkorištenijih struktura za pohranu hijerarhijski strukturiranih podataka.

Pretraživanje binarnog stabla većinom počinje iz korjena prema listovima. Najpoznatije dvije metode su:

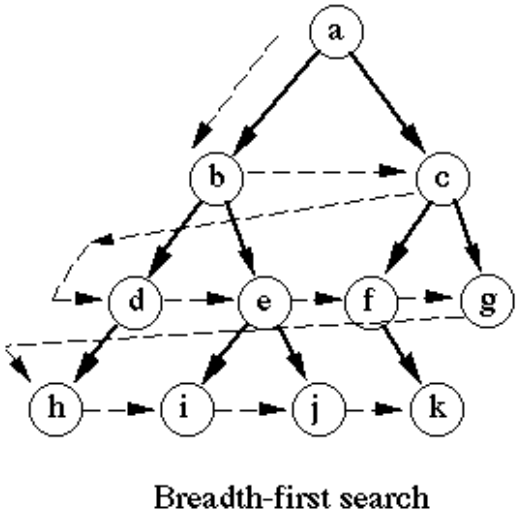
- Vertikalno pretraživanje (eng. *Depth-first search,* kratica: DFS)
- Horizontalno pretraživanje (eng. *Breadth-first search,* kratica: BFS)

Kod vertikalnog pretraživanja binarnog stabla kreće se od korjena stabla prema listovima. Cilj vertikalnog pretraživanja je istražiti cijeli put od korjena do lista nakon čega se vraća prema korijenu. Postupak se ponavlja sve dok nisu ispitane sve grane. Ovakav način najčešće zahtjeva zapisivanje podataka u stog prilikom pretraživanja binarnog stabla. Na slici 4. se može vidjeti slijed vertikalnog pretraživanja.



Slika 8. Vertikalno pretraživanje (Prema: Tim NG, 2014)

Horizontalno pretraživanje podrazumjeva pretraživanje binarnog stabla po razinama. Kreće se od korjena prema listovima stabla. Algoritam se spušta na nižu razinu tek kada je obuhvatio sve čvorove koji se nalaze na trenutnoj razini.



Slika 9. Horizontalno pretraživanje (Prema: Tim NG, 2014)

DFS i BFS su dva najraširenija algoritma za pretraživanje binarnog stabla. Postoji tri vrste DFS algoritma pretraživanja: PreOrder, InOrder i PostOrder. U ovom radu smo obuhvatili najraširenije i najkorišteniju verziju DFS PreOrder.

8. Rekurzivni upiti

Rekurzivni upiti služe za dohvaćanje hijerarhijski strukturiranih podataka te za ispis podataka po određenim razinama. U PostgreSQL jeziku za kreiranje rekurzivnog upita potrebna je ključna riječ RECURSIVE nakon WITH klauzule koja označava da će se nad stvorenom CTE tablicom izvršiti rekurzivni upit. Jedna od najvećih prednosti CTE-a je mogućnost izvršavanja rekurzivnih upita. Takvi upiti omogućuju dohvaćanje hijerarhijski strukturiranih podataka po razinama.

Rekurzivni upiti sadrže:

- Common Table Expression
- Sidreni upit
- UNION operator
- Rekurzivni upit

Jednostavni primjer rekurzivnog upita je:

```
WITH RECURSIVE CTE_tablica AS
(SELECT 1 AS broj
 UNION
 SELECT broj + 1
 FROM CTE_tablica
 WHERE broj < 5000)
SELECT broj FROM CTE_tablica;
```

Koristeći ključnu riječ RECURSIVE naglašavamo kako se nad CTE izvršava rekurzivni upit. U ovom primjeru stupci u CTE-u nisu navedeni što znači da se u CTE preslikavaju svi stupci iz sidrenog upita, u ovom primjeru to je jedan stupac (broj). Nakon toga slijedi UNION operator koji spaja i ispisuje sve dobivene rezultate. U rekurzivnom upitu se dohvaća prvi ispisani redak (s brojem 1) koji se inkrementira za 1 te ispisuje. Nakon prve iteracije u tablici se nalaze dva reda s brojevima 1 i 2. Ovaj postupak se ponavlja dok je uvjet naveden pod WHERE klauzulom zadovoljen. Nakon što uvjet završetka više nije zadovoljen petlja se prekida te se ispisuje rezultat (tablica s brojevima od 1 do 5000) [11] [12].

8.1. Rekurzivni upit nad jednom tablicom

Tablica koja ima vezu sa samom sobom je preduvjet za stvaranje rekurzivnih upita nad jednom tablicom. Kako bi pokazali stvarni primjer dohvaćanja hijerarhijskih podataka uz pomoć rekurzivnog upita generirana je hijerarhijska struktura uz pomoć online generatora „Mockaroo“ [13]. Podaci su generirani na način da svaki nadređeni ima 7 zaposlenika ispod sebe. Fiktivna organizacija ima 7 razina što sveukupno čini 137 257 zaposlenika. Da bi podaci bili hijerarhijski zapisani u tablicu kreiran je stupac id_sefa koji je povezan sa stupcem id iste tablice (samoreferencirajuća tablica). Glavni direktor nema nadređenog što znači da je njegov id_sefa = NULL. Sljedeći rekurzivni upit ispisuje sve zaposlenike fiktivne organizacije po razinama:

```
WITH RECURSIVE ZaposleniciCTE (ID, Prezime, Ime, Sef, razina) AS

(SELECT "zaposlenici"."id", "zaposlenici"."ime", "zaposlenici"."prezime",
"zaposlenici"."id_sefa", 1
FROM zavrsni."zaposlenici" WHERE "id_sefa" IS NULL

UNION

SELECT "zaposlenici"."id", "zaposlenici"."ime", "zaposlenici"."prezime",
"zaposlenici"."id_sefa", ZaposleniciCTE.razina + 1
FROM zavrsni."zaposlenici"
JOIN ZaposleniciCTE
ON zaposleniciCTE.ID = Zaposlenici.id_sefa)

SELECT CONCAT_WS(' ', EmpCTE.Ime, EmpCTE.Prezime) AS Zaposlenik,
CASE WHEN SefCTE.Ime IS NULL THEN 'Super Boss' ELSE SefCTE.Ime END AS Sef,
EmpCTE.razina
FROM ZaposleniciCTE EmpCTE
LEFT JOIN ZaposleniciCTE SefCTE
ON EmpCTE.Sef = SefCTE.ID
ORDER BY razina;
```

Upit je podjeljen na pet dijelova koji su međusobno odvojeni razmacima za lakše objašnjavanje i razumijevanje. U prvom dijelu SQL upita koristi se WITH klauzula nakon čega slijedi ključna riječ RECURSIVE kako bi PostgreSQL znao da se radi o rekurzivnom upitu. Nakon toga slijedi ime CTE nad kojim se upit izvršava te u zagradi navedeni stupci. Dakle prvi red zapravo označava ime i stupce CTE-a koji će se koristiti u upitu odmah nakon zagrada.

Drugi dio upita sadrži sidreni upit koji označava na kojem dijelu rekurzija počinje, u ovom slučaju je to glavni direktor fiktivne organizacije. Glavni direktor nema id_sefa zbog čega dohvaćamo sve redove iz tablice zaposlenici kojima je id_sefa jednak NULL vrijednosti. No ti podaci će se preslikati u CTE koji ima svoje zadane stupce. U ovom upitu želimo da se pokraj svakog zaposlenika pokaže u kojoj se razini hijerarhije taj zaposlenik nalazi. Zbog toga je kreiran stupac razina u CTE tablici. Kako znamo da se glavni direktor nalazi na samom vrhu

organizacijske hijerarhije, stavljamo broj 1 u stupac razina. Treći dio upita je operator UNION koji označava spajanje dva SELECT upita (u ovom slučaju spajanje sidrenog i rekurzivnog upita) i ispisivanje podataka iz oba upita.

Zadnji upit unutar WITH klauzule sadrži rekurziju te je četvrti dio SQL upita. U rekurzivnom upitu se prvo označavaju svi stupci koje je potrebno dohvatiti i preslikati u CTE tablicu. Zadnjem stupcu koji označava hijerarhijsku razinu se inkrementira vrijednost za 1 jer se svaku petlju upit spušta na nižu razinu hijerarhije. Nakon toga slijedi rekurzivni dio upita, dakle pomoću JOIN klauzule spajamo CTE tablicu sa samom sobom. Na taj način se dobivena tablica spaja sa samom sobom. Postupak se ponavlja sve dok nisu obuhvaćeni svi zaposlenici iz organizacije te se svi upisuju u CTE zbog UNION operatora.

Zatvorena zagrada označava kraj WITH klauzule, dakle kreirana je CTE tablica pomoću rekurzivnog upita koja se može upotrijebiti u samo jednom sljedećem upitu prije nego je trajno izbrisana iz memorije. Nakon što je kreirana CTE tablica svi dohvaćeni podaci se mogu ispisati jednostavnim SELECT upitom no takvi podaci nisu formatirani te su teško čitljivi. Pogledati sliku 10. koja predstavlja rezultat ispisa CTE tablice sa sljedećim SELECT upitom:

```
SELECT * FROM ZaposleniciCTE;
```

	id	prezime	ime	sef	razina
1	1	Cudmore	Emerson	<null>	1
2	2	Letford	Liva	1	2
3	3	Fanthom	Othelia	1	2
4	4	Benting	Hoyt	1	2
5	5	Pyvis	Izzy	1	2
6	6	Fearon	Irma	1	2
7	7	Mithun	Rozalie	1	2
8	8	Tookey	Loy	1	2
9	9	Johnathan	Cornuau	2	3
10	10	Denny	Giuroni	2	3
11	11	Rafa	Abelson	2	3
12	12	Demetrius	Ashburne	5	3
13	13	Lefty	Kupisz	4	3
14	14	Mano	Casillas	7	3
15	15	Shaine	Colvie	6	3
16	16	Kordula	Lyptrit	5	3
17	17	Maurice	Leverton	3	3
18	18	Devon	Flanders	3	3
19	19	Nadiya	Elton	7	3
20	20	Elwyn	New	4	3
21	21	Willie	Andresen	7	3

Slika 10. Neformatirani rezultat rekurzivnog upita (autorski rad)

Takav rezultat je u potpunosti ispravan te ispisuje sve potrebne podatke no teško je čitljiv. Upravo iz tog razloga se unutar zadnje SELECT klauzule koriste ugrađene PostgreSQL funkcije kako bi ispis dobivenih podataka bio jednostavniji za čitanje. CONCAT_WS funkcija služi za spajanje dva stupca u jedan sa separatorom (eng. *Concatenation_WithSeparator*). Uz pomoć ove funkcije spajamo ime i prezime u jedan stupac. Unutar navodnika se prvo navodi separator što je u našem slučaju razmak, nakon čega slijede stupci koji predstavljaju ime i prezima svakog zaposlenika te se na kraju tom stupcu se zadaje ime „Zaposlenik“. Također se modificira stupac za ispis šefa gdje se za svakog zaposlenika ispisuje ime njegovog nadređenog. Ovdje se mora uzeti u obzir glavni direktor koji nema nadređenog, dakle u tom će se slučaju ispisati NULL vrijednost što nije intuitivno. Iz tog razloga se koristi CASE WHEN izraz koji omogućuje ispis podataka ovisno o određenom uvjetu, u ovom slučaju uvjet je NULL vrijednost. Dakle ukoliko je id_sefa jednak NULL vrijednost ispisuje se „Super Boss“, dok se u svim ostalim slučajevima ispisuje ime šefa. Nakon završetka CASE WHEN izraza slijedi riječ END kako bi PostgreSQL znao gdje je kraj. Taj stupac predstavlja šefa zbog čega mu se zadaje ime „Sef“.

Kako bi ispisali ime šefa tablica se mora referencirati na samu sebe (zbog dohvaćanja imena prema id_sefa). No u ovom slučaju je potrebno koristiti LEFT JOIN kako bi svi podaci bili dohvaćeni (ukoliko se koristi JOIN operator glavni direktor neće biti ispisan). Na kraju je potrebno sortirati podatke po hijerarhijskim razinama prema BFS algoritmu zbog čega je i stvoren stupac „razina“.

	zaposlenik	sef	razina
1	Emerson Cudmore	Super Boss	1
2	Izzy Pyvis	Emerson	2
3	Hoyt Benting	Emerson	2
4	Loy Tookey	Emerson	2
5	Irma Fearon	Emerson	2
6	Liva Letford	Emerson	2
7	Rozalie Mithun	Emerson	2
8	Othelia Fanthom	Emerson	2
9	Pleace Wenonah	Liva	3
10	Abelson Rafa	Liva	3
11	Giuroni Denny	Liva	3
12	Noyes Temp	Liva	3
13	Dinzey Jada	Othelia	3
14	Biasioni Mattias	Othelia	3
15	Dobeson Obadias	Othelia	3
16	Flanders Devon	Othelia	3
17	Jeannard Tammy	Liva	3
18	Caldes Wat	Liva	3
19	Sturt Kala	Othelia	3
20	Culpin Mahmud	Othelia	3

Slika 11. Rezultat rekurzivnog SQL upita (autorski rad)

8.2. Rekurzivni upit nad dvije tablice

Najčešći primjer upotrebe rekurzivnih upita s dvije tablice u praksi je kod resporeda materijala (eng. *Bill of Materials,* kratica: BOM). BOM je popis proizvoda te svih materijala od kojih je pojedini proizvod sastavljen. Na taj način su zapisani svi materijali te se može ustanoviti koji materijali su potrebni za koji proizvod. Tablice potrebne za izvođenje rekurzivnog upita u svrhu dohvaćanja i ispisivanja rasporeda materijala su preuzete iz baze podataka „AdventureWorks2016.bak“ od proizvođača bicikala „AdventureWorks„.[14] Kako bi lakše razumjeli rekurzivni upit te razlog njegove primjene prvo trebamo objasniti način zapisa podataka u bazu podataka. Dakle bicikl je sastavljen od kotača, kočnica, lanca, sjedala itd. No nijedan od tih materijala nije zadnji u proizvodnom ciklusu, dakle, kotač se sastoji od unutarnje gume, vanjske gume, osovine, dok se svaki od tih artikala nadalje sastoji od nekoliko materijala niže proizvodne razine. Na neki način se BOM može gledati kao hijerarhijska struktura u kojoj je na vrhu završni proizvod (u ovom slučaju bicikl) dok su na dnu hijerarhije sirovine potrebne za izradu najjednostavnijih materijala.

Ukoliko firma proizvodi bicikle mora imati sve artikle i materijale potrebne za njihovu proizvodnju u svojoj bazi podataka. No također mora imati uvid u popis materijala potrebnih za izradu pojedinog artikla (ukoliko sama proizvodi određene artikle kako bi uštedjela na proizvodnji). U takvom slučaju se stvaraju dvije tablice, u jednoj tablici se nalaze svi proizvodi (od bicikla do početnih sirovina) dok druga tablica služi kao pomoćna BOM tablica koja povezuje sve artikle u hijerarhijsku strukturu. Takva tablica povezuje artikl više razine sa svim materijalima koji su potrebni za njegovi izradu uz eventualne dodatne podatke vezane uz pojedini artikl (npr. Količina materijala potrebna za izradu artikla). Upravo iz razloga što BOM tablica sadrži artikl i materijal postoje dvije veze iz BOM tablice u tablicu „proizvodi“ jer su i artikl i materijali potrebni za izradu tog artikla sadržani u tablici proizvoda. Tako su svi podaci strukturirani u bazi podataka na način da se za svaki artikl može ispisati popis materijala potrebnih za njegovu proizvodnju.

No takav način zapisa podataka stvara komplikacije kod ispisa svih proizvoda uz popis materijala od kojih je isti sastavljen. Upit potreban za ispis podataka na taj način se može napisati iterativnom i rekurzivnom metodom. Ukoliko se ispisuje na iterativni način za svaku razinu je potrebno napisati podupit kako bi se dohvatile sve razine proizvodnje. Dakle ukoliko se u međuvremenu stvori nova razina proizvodnje, upit se mora nadograđivati s novim podupitom kako bi se dohvatila još jedna razina proizvodnje što stvara velike komplikacije.

Korištenjem rekurzije se uklanja potreba za pisanjem podupita za svaku razinu proizvodnje što uvelike pojednostavnjuje upit. To je upravo iz razloga što se rekurzija postavlja na način da dohvaća sve podatke iz iste razine proizvodnje nakon čega se „spušta“ na nižu razinu dok ne dohvati sve materijale iz tablice proizvoda.

Dolje navedeni upit dohvaća sve proizvode te ih sortira na način da ispisuje artikl nakon čega slijedi popis proizvoda od kojih je isti sastavljen:

```
WITH RECURSIVE bomCTE (id_proizvoda, naziv, boja, kolicina, razina,
product_assembly_id, sort) AS
(SELECT p.product_id, p.name, p.color, 1, 1, NULL, p.name
FROM zavrsni."product" AS p JOIN zavrsni."bom"
ON bom.component_id = p.product_id
AND bom.product_assembly_id IS NULL

  UNION

SELECT p.product_id, p.name, p.color, bom.per_assembly_qty, bomCTE.razina +
1,
  CAST(bomCTE.id_proizvoda AS TEXT), CAST(CONCAT_WS('/', bomCTE.sort,
p.name) AS varchar(100))
  FROM bomCTE JOIN zavrsni.bom
    ON bom.product_assembly_id = bomCTE.id_proizvoda
    JOIN zavrsni.product as p
    ON bom.component_id = p.product_id)

SELECT bomCTE.naziv, CASE WHEN bomCTE.boja IS NULL THEN 'None' ELSE
bomCTE.boja END AS Boja, bomCTE.kolicina, bomCTE.razina, bomCTE.sort FROM
bomCTE
ORDER BY sort;
```

Upit je napravljen prema primjeru K. Wenzel-a [15]. Složeni upit je podjeljen na istih pet dijelova kao i rekurzivni upit nad jednom tablicom. Dakle prvi dio služi za kreiranje CTE tablice sa svim potrebnim stupcima. Ovdje je važno napomenuti stupac „sort“ koji služi za sortiranje artikala po razinama. Stupac količina predstavlja količinu potrebnu za proizvodnju artikla za razinu iznad. U taj stupac će se preslikavati „per_assembly_quantity“ stupac iz BOM tablice. No iz razloga što navedeni stupac ne postoji u tablici proizvoda odakle se dohvaćaju podaci u sidrenom upitu na tom mjestu se nalazi broj 1 (bicikl je najviša razina hijerarhije).

Drugi dio upita je sidreni upit koji dohvaća sve artikle iz tablice proizvodi koji se nalaze na vrhu hijerarhije, dakle sve artikle koji nisu sastavni dio nijednog drugog artikla. U ovom slučaju to su svi bicikli jer je bicikl krajnji proizvod odabrane proizvodne linije. Prilikom pisanja sidrenog upita mora se voditi računa o stupcima koji se označuju jer se rezultat preslikava u stupce CTE tablice. Iz tog razloga se na pozicije za količinu i razinu stavljaju brojevi 1 (prva razina proizvodnje). Stupac „product_assembly_id“ je NULL jer taj stupac predstavlja proizvod koji je razinu iznad od trenutno promatranog proizvoda (ukoliko je vrijednost NULL proizvod je

na najvišoj razini – bicikl). No u tablici proizvoda nema podatak o tome koji proizvod je na vrhu hijerarhije zbog čega se mora spojiti s BOM tablicom koristeći JOIN. Nakon spajanja slijedi ključni uvijet u kojem se nakon spajanja provjeravaju i dohvaćaju samo oni artikli iz tablice proizvoda koji u BOM tablici imaju NULL vrijednost u stupcu „product_assembly_id“.

Nakon što su dohvaćene bazne vrijednosti koristeći sidreni upit slijedi UNION operator koji spaja sidreni i rekurzivni upit. U rekurzivnom upitu se dohvaćaju svi potrebni podaci iz BOM tablice i tablice proizvoda dok se broj hijerarhijske razine povećava ovisno o razini hijerarhije koja se dohvaća u tom trenutku. U ovom slučaju prilikom stvaranja stupaca za CTE tablicu nisu eksplicitno određeni tipovi podataka zbog čega stupci poprimaju tip podatka iz sidrenog upita. Ukoliko se tipovi podataka iz sidrenog i rekurzivnog upita ne poklapaju PostgreSQL javlja grešku. Vrijednost NULL se mapira u stupac tipa TEXT. Upravo zbog toga je potrebno id_proizvoda pretvoriti u TEXT. Stupac je ključan za ispis artikala po razinama. Upit funkcionira na način da prvo dohvati sve proizvode na vrhu hijerarhijske strukture (bicikle), nakon čega od svakog bicikla dohvaća sve njihove djelove. Ukoliko ne bismo imali stupac sort ne bi bilo jednostavnog načina za ispis svih materijala od kojih je bicikl sastavljen odmah ispod.

Kada su dohvaćeni svi bicikli pretražuju se materijali od kojih se sastoje ti bicikli. Svaki pronađeni artikl će u stupcu sort imati zapisano bicikl/artikl. U trećoj razini svi materijali će u sortu imati zapisano bicikl/artikl/materijal. Ovaj postupak se nastavlja do zadnje sirovine koja se koristi u proizvodnji. Na taj način je u stupcu sort zapisano koji je proizvodni slijed svakog materijala. U sidrenom upitu kao prvi zapis je dohvaćen naziv proizvoda (bicikla) koji je tipa varchar(100). Zbog toga se svi podaci koji se zapisuju u stupac sort moraju pretvoriti u tip varchar. Također se koristi funkcija CONCAT_WS objašnjena ranije u radu kako bi nakon se nakon svake razine dodao novi proizvod. Dakle bomCTE.sort je prošla vrijednost na koju se dodaje naziv trenutnog proizvoda sa separatorom „/“. Nakon što su navedeni svi stupci i pretvoreni u pravi tip spajaju se tablica proizvoda i BOM tablica s CTE tablicom jer se u upitu koriste vrijednosti iz sve tri tablice.

Formatiranje CTE tablice je zadnji korak u rekurzivnom upitu. U ovom primjeru je potrebno ispisati podatke po DFS algoritmu. Zbog tog razloga je stvoren stupac sort koji ima zapisane sve čvorove od lista do korjena. Rezultat sortiranja po takvom stupcu je popis proizvoda sa svim materijalima potrebnim za njegovu proizvodnju. Također postoji još nekoliko sitnica koje se mogu prepraviti kako bi ispisana tablica bila jednostavnija za pročitati. Formatiranje rezultata nije obavezno no ukoliko se rezultat ne formatira ispisuju se svi proizvodi prve razine, nakon čega slijede svi proizvodi druge razine itd. Takav ispis ne daje informacije koje se očekuju od ispisivanja BOM-a. Neformatirani rezultat rekurzivnog upita se može vidjeti ukoliko se nakon WITH klauzule ispiše cijela CTE tablica (SELECT * FROM bomCTE).

	id_proizvoda	naziv	boja	kolicina	razina	product_assembly_id	sort
1	750	Road-150 Red, 44	Red	1	1	<null>	Road-150 Red, 44
2	750	Road-150 Red, 44	Red	1	1	<null>	Road-150 Red, 44
3	751	Road-150 Red, 48	Red	1	1	<null>	Road-150 Red, 48
4	752	Road-150 Red, 52	Red	1	1	<null>	Road-150 Red, 52
5	753	Road-150 Red, 56	Red	1	1	<null>	Road-150 Red, 56
6	753	Road-150 Red, 56	Red	1	1	<null>	Road-150 Red, 56
7	754	Road-450 Red, 58	Red	1	1	<null>	Road-450 Red, 58
8	755	Road-450 Red, 60	Red	1	1	<null>	Road-450 Red, 60
9	756	Road-450 Red, 44	Red	1	1	<null>	Road-450 Red, 44
10	757	Road-450 Red, 48	Red	1	1	<null>	Road-450 Red, 48
11	758	Road-450 Red, 52	Red	1	1	<null>	Road-450 Red, 52
12	759	Road-650 Red, 58	Red	1	1	<null>	Road-650 Red, 58
13	760	Road-650 Red, 60	Red	1	1	<null>	Road-650 Red, 60
14	761	Road-650 Red, 62	Red	1	1	<null>	Road-650 Red, 62
15	762	Road-650 Red, 44	Red	1	1	<null>	Road-650 Red, 44
16	763	Road-650 Red, 48	Red	1	1	<null>	Road-650 Red, 48
17	764	Road-650 Red, 52	Red	1	1	<null>	Road-650 Red, 52

Slika 12. Neformatirani ispis rasporeda materijala (autorski rad)

Kod formatiranja upita ne uključujemo id proizvoda koji je u ovom primjeru nepotreban. Neki proizvodi imaju oznaku u kojoj se boji određeni artikl proizvodi dok drugi nemaju zbog čega se koristi CASE WHEN izraz. Ako određeni materijal nema određenu boju ispisuje se „None“, inače se ispisuje zadana boja. Nakon toga se označuju svi proizvodi koji su potrebni za ispis te se sortira po stupcu sort.

	naziv	boja	kolicina	razina	sort
1	Mountain-100 Black, 38	Black	1	1	Mountain-100 Black, 38
2	Chain	Silver	1	2	Mountain-100 Black, 38/Chain
3	Front Brakes	Silver	1	2	Mountain-100 Black, 38/Front Brakes
4	Front Derailleur	Silver	1	2	Mountain-100 Black, 38/Front Derailleur
5	Front Derailleur Cage	Silver	1	3	Mountain-100 Black, 38/Front Derailleur/Front Derailleur Cage
6	Front Derailleur Linkage	Silver	1	3	Mountain-100 Black, 38/Front Derailleur/Front Derailleur Linkage
7	HL Bottom Bracket	None	1	2	Mountain-100 Black, 38/HL Bottom Bracket
8	BB Ball Bearing	None	10	3	Mountain-100 Black, 38/HL Bottom Bracket/BB Ball Bearing
9	Bearing Ball	None	10	4	Mountain-100 Black, 38/HL Bottom Bracket/BB Ball Bearing/Bearing Ball
10	Cone-Shaped Race	None	2	4	Mountain-100 Black, 38/HL Bottom Bracket/BB Ball Bearing/Cone-Shaped Race
11	Cup-Shaped Race	None	2	4	Mountain-100 Black, 38/HL Bottom Bracket/BB Ball Bearing/Cup-Shaped Race
12	Lock Ring	Silver	1	4	Mountain-100 Black, 38/HL Bottom Bracket/BB Ball Bearing/Lock Ring
13	HL Shell	None	1	3	Mountain-100 Black, 38/HL Bottom Bracket/HL Shell
14	HL Crankset	Black	1	2	Mountain-100 Black, 38/HL Crankset
15	Chainring	Black	3	3	Mountain-100 Black, 38/HL Crankset/Chainring
16	Chainring Bolts	Silver	3	3	Mountain-100 Black, 38/HL Crankset/Chainring Bolts
17	Chainring Nut	Silver	3	3	Mountain-100 Black, 38/HL Crankset/Chainring Nut
18	Freewheel	Silver	1	3	Mountain-100 Black, 38/HL Crankset/Freewheel
19	HL Crankarm	Black	2	3	Mountain-100 Black, 38/HL Crankset/HL Crankarm
20	HL Headset	None	2	2	Mountain-100 Black, 38/HL Headset

Slika 13. Ispis rasporeda materijala pomoću rekurzivnog upita (autorski rad)

8.3. Usporedba iterativnog i rekurzivnog načina

Dohvaćanje podataka iz dvije tablice može se izvršiti na dva načina: iterativno i rekurzivno. Iterativni način je dohvaćanje podataka korištenjem podupita nakon čega se sve razine spajaju s UNION operatorom dok se za rekurzivni način u PostgreSQL-u koristi klauzula WITH RECURSIVE. Ispis podataka uz pomoć rekurzije je relativno kompleksno te se ne preporuča ukoliko se radi o malom skupu podataka. S druge strane ukoliko se radi o velikom skupu podataka koji je organiziran po razinama, korištenje rekurzije može znatno pojednostaviti pisanje upita. Pretpostavimo kako se korištenjem rekurzije želi doći do svih podataka, dakle ispisati sve hijerarhijske razine. Kako bi došli do najrealnijih rezultata kod uspoređivanja brzine stvorene su zasebne tablice ovisno o broju hijerarhijskih razina. Razlog tomu je pretpostavka da se uvijek dohvaćaju svi podaci iz baze podataka. Za svrhu testiranja brzine između iterativnog i rekurzivnog načina dohvaćanja podataka koristit ćemo 8 tablica. Podaci u svim tablicama su generirani pomoću online generatora „Mockaroo“. Prve 4 tablice su generirane na način da svaki nadređeni ima 7 podređenih zaposlenika ispod sebe u hijerarhiji. Tablice su redom: „zaposlenici3“, „zaposlenici4“, „zaposlenici5“, „zaposlenici6“ te se razlikuju u dubinama hijerarhije koju predstavlja broj unutar naziva tablice. U druge 4 tablice nadređeni ima samo dva podređena zaposlenika no u ovom slučaju fiktivna organizacijska struktura ima više hijerarhijskih razina. Nazivi tablice su: „zaposlenici10“, „zaposlenici11“, „zaposlenici12“, „zaposlenici13“ te imaju 10, 11, 12 i 13 hijerarhijskih razina respektivno.

Dakle testiraju se dvije fiktivne organizacije na četiri hijerarhijske razine. Prva organizacija ima široku hijerarhijsku strukturu s većim brojem zaposlenika i manjem broju razina dok druga organizacija ima duboku hijerarhijsku strukturu s relativno niskim brojem zaposlenika na velik broj hijerarhijskih razina. Na ovaj način ćemo testirati uspješnost rekurzivnog upita te ga usporediti s iterativnim načinom dohvaćanja podataka. Zbog kompleksnosti i opširnosti iterativnog SQL upita u ovom radu ćemo obraditi primjer dohvaćanja upita s tri hijerarhijske razine. Nakon navedenog i objašnjenog primjera oba upita ispitati ćemo njihovu uspješnost te zaključiti u kojim situacijama je preporučeno koristiti rekurzivne upite.

Dolje napisan SQL upit se koristi za sve tablice i sve hijerarhijske razine, jedina razlika u upitima je izmjena naziva tablice iz koje se podaci dohvaćaju. Upravo to je najveća prednost rekurzivnih upita u odnosu na iterativne upite.

```
WITH RECURSIVE ZaposleniciCTE (ID, Prezime, Ime, Sef, razina) AS
(SELECT z."id", z."ime", z."prezime", z."id_sefa", 0
FROM zavrsni."zaposlenici6" z WHERE "id_sefa" IS NULL

UNION

SELECT z."id", z."ime", z."prezime", z."id_sefa", ZaposleniciCTE.razina + 1
FROM zavrsni."zaposlenici6" z
JOIN ZaposleniciCTE
ON zaposleniciCTE.ID = z.id_sefa)

SELECT * FROM ZaposleniciCTE
ORDER BY razina;
```

137257 rows

	id	prezime	ime	sef	razina
1	1	Cudmore	Emerson	<null>	0
2	2	Letford	Liva	1	1
3	4	Benting	Hoyt	1	1
4	8	Tookey	Loy	1	1
5	5	Pyvis	Izzy	1	1
6	3	Fanthom	Othelia	1	1
7	6	Fearon	Irma	1	1
8	7	Mithun	Rozalie	1	1
9	16	Kordula	Lyptrit	5	2
10	32	Constancy	Dumingo	6	2
11	33	Frankie	Fawcitt	8	2

Slika 14. Rezultat rekurzivnog upita (autorski rad)

Dakle uspoređuju se brzine izvršavanja rekurzivnog i iterativnog upita nad dvije organizacijske strukture. Prva organizacijska struktura ima šest razina hijerarhije i 137 257 zaposlenika koju ćemo zvati organizacija „A“. S druge strane imamo organizaciju „B“ koja ima 13 razina hijerarhije te 16 383 zaposlenika. U ovom istraživanju top razina hijerarhije ima jednog zaposlenika koji čini nultu razinu hijerarhije.

Istraživanja se vrše na prijenosnom prijenosnom računalu Lenovo Yoga 520. Prijenosno računalo je opremljeno s Intel Core i7-7500U procesorom snage 2.70GHz, Nvidia GeForce 940MX grafičkom karticom te 8GB RAM-a. Sva mjerenja su vršena u DataGrip sustavu za upravljanje bazama podataka. Svaki podatak je prosjek uzorka od 10 pojedinačnih mjerenja. Svi podaci istraživanja su priložena uz rad.

Pisanje iterativnog SQL upita koji dohvaća podatke o zaposlenicima po hijerarhijskim razinama zahtjeva pisanje podupita za svaku hijerarhijsku razinu. Na primjeru možemo vidjeti dohvaćanje podataka o zaposlenicima u organizaciji koja se sadrži tri hijerarhijske razine:

```
SELECT z.id, z.ime, z.prezime, z.id_sefa
FROM   "zavrsni".zaposlenici6 z
  WHERE id_sefa IS NULL

UNION

SELECT z.id, z.ime, z.prezime, z.id_sefa
FROM   "zavrsni".zaposlenici6 z
       JOIN "zavrsni".zaposlenici6 sef
       ON z.id_sefa= sef.id
       AND z.id_sefa IN

                  (SELECT z.id
                  FROM   "zavrsni".zaposlenici6 z
                    WHERE z.id_sefa IS NULL
                  )

UNION

SELECT z.id, z.ime, z.prezime, z.id_sefa
FROM   "zavrsni".zaposlenici6 z
       JOIN "zavrsni".zaposlenici6 sef
       ON z.id_sefa= sef.id
       AND z.id_sefa IN

           (SELECT z.id
              FROM   "zavrsni".zaposlenici6 z
                JOIN "zavrsni".zaposlenici6 sef
                ON z.id_sefa= sef.id
                AND z.id_sefa IN

                  (SELECT z.id
                  FROM   "zavrsni".zaposlenici6 z
                    WHERE z.id_sefa IS NULL
                  ))

UNION

SELECT z.id, z.ime, z.prezime, z.id_sefa
FROM   "zavrsni".zaposlenici6 z
       JOIN "zavrsni".zaposlenici6 sef
       ON z.id_sefa= sef.id
       AND z.id_sefa IN

           (SELECT z.id
              FROM   "zavrsni".zaposlenici6 z
                JOIN "zavrsni".zaposlenici6 sef
                ON z.id_sefa= sef.id
                AND z.id_sefa IN

                  (SELECT z.id
              FROM   "zavrsni".zaposlenici6 z
                JOIN "zavrsni".zaposlenici6 sef
                ON z.id_sefa= sef.id
                AND z.id_sefa IN

                  (SELECT z.id
                  FROM   "zavrsni".zaposlenici6 z
                    WHERE z.id_sefa IS NULL
                  )));
```

Na primjeru je vidljiv velik nedostatak pisanja upita za dohvaćanje podataka korištenjem iterativnog načina. Upit se sastoji od glavne četiri SELECT klauzule koje dohvaćaju nultu i prve tri hijerarhijske razine dok se između njih nalazi UNION operator koji spaja sve dobivene rezultate. Dakle u prvom SELECT upitu se dohvaćaju svi zaposlenici iz tablice „zaposlenici6“ kojima je id_sefa NULL vrijednost (top razina hijerarhije). U drugom upitu je potrebno spojiti tablicu sa samom sobom kako bi dobili sve zaposlenike prve razine. Upit vraća sve zaposlenike kojima je id_sefa jednak prvom SELECT upitu (dakle id_sefa = NULL). Napisana dva upita se spajaju UNION operatorom te vraćaju zajedno vraćaju nultu i prvu razinu hijerarhije. Dohvaćanjem druge razine hijerarhije se koristi ista tehnika, dakle SELECT klauzula vraća popis svih zaposlenika koji imaju id_sefa jednak drugom upitu (svi zaposlenici kojima su nadređeni zaposlenici prve hijerarhijske razine). Postupak se ponavlja dok se ne obuhvate sve potrebne hijerarhijske razine. Dakle ukoliko struktura podataka sadrži 13 razina potrebno je napisati 14 SELECT klauzula (uključujući nultu razinu) te ih sve spojiti s UNION operatorom. Takav upit sadrži 630 redova što ovu metodu čini uvelike kompliciranijom ukoliko se radi o takvom skupu podataka.

Prilikom uspoređivanja brzine obrade upita bilježile su se tri vrijednosti. Prva vrijednost predstavlja ukupno vrijeme potrebno za izvršavanje upita i dohvaćanje podataka. Druga vrijednost je vrijeme potrebno za samo izvršavanje upita dok treća vrijednost sadrži vrijeme potrebno za dohvaćanje podataka. Svi podaci su zabilježeni u milisekundama (ms) od kojih su sastavljene tri tablice za usporedbu dobivenih podataka. U prvoj tablici se uspoređuje ukupno potrebno vrijeme obrade i dohvaćanja podataka između rekurzivnog i iterativnog upita. Druga tablica uspoređuje vrijeme izvršavanja upita, a treća vrijeme potrebno za dohvaćanje podataka. Na taj način se mogu vidjeti svi prednosti i nedostaci obje vrste upita.

Tablica 1. Usporedba ukupnog vremena obrade upita organizacije „A“

Broj razina	Skup podataka	Rekurzija (u ms)	Iteracija (u ms)	Uspješnost rekurzije
3	400	21,9	33,7	53,88%
4	2801	32,7	50,4	54,13%
5	19608	135,4	175,4	29,54%
6	137257	1245,7	1399,3	12,33%
			Prosječno:	37,47%

Mjerenje ukupnog vremena potrebnog za obradu upita na organizaciji „A“ rezultiralo je prosječnoj uspješnosti rekurzivnog upita nad iterativnim upitom od 37.47%. No vrijedi napomenuti kako se uspješnost rekurzije smanjuje nakon svakog povećanja broja hijerarhijske razine za prosječno 13.85%. Razlog tomu je široka hijerarhijska struktura, dakle velik broj podataka na mali broj hijerarhijskih razina.

Tablica 2. Usporedba vremena izvršavanja upita organizacije „A“

Broj razina	Skup podataka	Rekurzija (u ms)	Iteracija (u ms)	Uspješnost rekurzije
3	400	9,9	18,1	82,82%
4	2801	14.2	27,3	92,54%
5	19608	56,0	94,6	68,93%
6	137257	578,7	859,2	48,47%
			Prosječno:	73,19%

Ukoliko se uzme u obzir samo vrijeme potrebno za izvršavanje upita uspješnost rekurzivnog upita se povećava na 73.19%. Također se može primjetiti blagi porast uspješnosti rekurzije nakon čega se nastavlja trend smanjenja uspješnosti od prosječno 11.45% po hijerarhijskoj razini. Najveća prednost rekurzivnih upita je upravo u vremenu izvršavanja i opsegu samog upita. Iako rezultati pokazuju kako su rekurzivni upiti brži i efikasniji od iterativnih upita trend smanjenja pokazuje na mogućnost točke u kojoj će iterativni način postati brži.

Tablica 3. Usporedba vremena dohvaćanja podataka organizacije „A“

Broj razina	Skup podataka	Rekurzija (u ms)	Iteracija (u ms)	Uspješnost rekurzije
3	400	12,0	15,6	30.00%
4	2801	18,5	23,1	24,86%
5	19608	79,4	80,8	1,76%
6	137257	667,0	540,1	-19,03%
			Prosječno:	9,40%

Usporedbom vremena za dohvaćanje podataka se ističu nedostaci rekurzivnih upita. Ukoliko promatrani podaci rastu eksponencijalno ovisno o razinama, uspješnost rekurzivnih upita će rasti do određene razine nakon čega uspješnost drastično pada zbog velikog opsega podataka. U ovom primjeru smanjenje uspješnosti rekurzije se očituje u petoj hijerarhijskoj razini. Dobiveni rezultati testiranja uspješnosti u organizaciji „A“ pokazuju kako su rekurzivni upiti brži i efikasniji način dohvaćanja podataka. No eksponencijalnim povećanjem broja podataka uspješnost rekurzije pada sve dok iterativni način ne postane brži. No postavlja se pitanje isplativosti pisanja iterativnog upita koji je daleko opsežniji i kompliciraniji ukoliko dođe do promjene u bazi podataka.

No unatoč dobivenim rezultatima na organizaciji „A“ potrebno je ispitati uspješnost rekurzije na organizaciji koja ima veći broj hijerarhijskih razina i relativno manji broj podataka. U takvim slučajevima se primjena rekurzije najčešće isplati zbog velike kompliciranosti pisanja iterativnog upita koji dohvaća sve razine. Isplativost rekurzije se očituje i u broju podataka koji treba dohvatiti jer je dohvaćanje velikog broja podataka smanjuje uspješnost rekurzije.

Tablica 4. Usporedba ukupnog vremena obrade upita organizacije „B“

Broj razina	Skup podataka	Rekurzija (u ms)	Iteracija (u ms)	Uspješnost rekurzije
10	2047	35,2	81,5	131,53%
11	4095	44,6	118,8	166,37%
12	8191	71,3	222,6	212,20%
13	16383	131,2	524,0	299,40%
			Prosječno:	202,38%

U ovom primjeru se može vidjeti velika razlika u uspješnosti rekurzije u odnosu na rezultate dobivene kod testiranja organizacije „A“. Također se može primjetiti kako se trend smanjenja uspješnosti rekurzije pretvorio u trend rasta od prosječno 55.96% po hijerarhijskoj razini. Razlog velikih razlika u odnosu na prošli primjer je povećanje broja hijerarhijskih razina te smanjenje opsega podataka koji se dohvaćaju što povećava ukupno vrijeme obrade rekurzivnih upita. Prosječna uspješnost rekurzije se povećala s 37.47% na 202.38% što je drastično velik postotak povećanja uspješnosti.

Tablica 5. Usporedba vremena izvršavanja upita na organizacije „B“

Broj razina	Skup podataka	Rekurzija (u ms)	Iteracija (u ms)	Uspješnost rekurzije
10	2047	14,1	58,8	317,02%
11	4095	21,2	87,8	314,15%
12	8191	33,8	166,6	392,90%
13	16383	60,8	443,8	629,93%
			Prosječno:	413,50%

Usporedbom samo vremena potrebnog za izvršavanje upita nad hijerarhijskom strukturom organizacije „B“ dolazi se do rezultata u kojima je rekurzivni upit neusporedivo brži. Rekurzivni upit se izvršava prosječno 413.50% brže od iterativnog upita uz konstantni trend rasta uspješnosti.

Tablica 6. Usporedba vremena dohvaćanja podataka organizacije „B“

Broj razina	Skup podataka	Rekurzija (u ms)	Iteracija (u ms)	Uspješnost rekurzije
10	2047	21,1	22,7	7,58%
11	4095	23,4	31,0	32,48%
12	8191	37,5	56,0	49,33%
13	16383	70,4	80,2	13,92%
			Prosječno:	25,83%

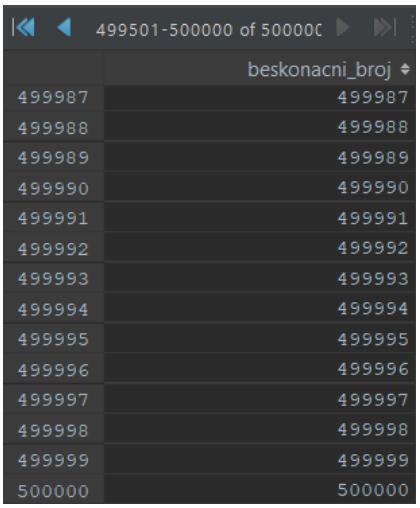
Kod usporedbe vremena dohvaćanja podataka dolazi se do zanimljivih rezultata. Ukoliko se usporede točke prijelaza iz trenda rasta i trenda smanjenja uspješnosti rekurzije u svim tablicama dolazimo do zaključka da uspješnost rekurzivnih upita počne opadati ukoliko se prosječni broj podataka po razinama nalazi između 700 i 1260 zbog drastičnog usporavanja kod dohvaćanja podataka. Dakle pisanje i eventualna izmjena rekurzivnih upita je daleko jednostavnija od iterativnog upita ukoliko se radi o dohvaćanju podataka iz više razina. Rekurzivni upiti su u prednosti i po pitanju brzine izvršavanja upita ukoliko se radi o manjem skupu podataka. Što je promatrani skup podataka veći od 700 podataka po razini uz više od 10 000 podataka to je veća mogućnost da će iterativni način dohvaćanja podataka biti brži od rekurzivnog upita.

8.4. Beskonačna rekurzija

Kod rekurzivnih upita potrebno je obratiti pažnju na podatke koji se ispisuju te ispitati postoji li mogućnost u kojem rekurzija nema točke završetka. Razni sustavi za upravljanje bazom podataka reagiraju različito kod detekcije beskonačne rekurzije. U tom slučaju MySQL javlja grešku dok PostgreSQL pokušava razrješiti grešku te ispisuje rezultate. Dakle PostgreSQL daje programerima više slobode kod pisanja rekurzivnih upita uz što dolazi i odgovornost kod pisanja takvih upita. Najjednostavniji primjer beskonačnog rekurzivnog upita je zbrajanje brojeva:

```
WITH RECURSIVE cte AS
(SELECT 1 AS n
 UNION
 SELECT n + 1
 FROM cte)
SELECT n AS beskonacni_broj FROM cte
LIMIT 500000;
```

Prilikom pokretanja rekurzivnog upita PostgreSQL ga provjerava te ukoliko postoji greška u upitu koja uzrokuje beskonačnu petlju pokušava ju razriješiti te ispisuje rezultat. Gore navedeni upit je primjer gdje može doći do beskonačne rekurzije. U sidrenom upitu definiramo naziv stupca „n“ te za prvu vrijednost postavljamo broj 1. Nakon toga se u rekurzivnom upitu broj n povečava za 1. Ovakav upit rezultira beskonačnom rekurzijom jer ne postoji uvjet završetka (npr. WHERE n < 5000). U ovom je slučaju na kraju upita postavljen LIMIT na 500000, dakle PostgreSQL ispisuje prvih 500000 podataka iz navedenog upita. Ako se LIMIT ukloni PostgreSQL će ući u beskonačnu petlju dok god se ne zaustavi izvršavanje upita.



Slika 15. Ispis beskonačnog broja (autorski rad)

9. Zaključak

Pojam rekurzije i rekurzivnih upita se na prvi pogled može činiti jako komplicirano zbog težeg razumijevanja samog pojma rekurzije. Nakon što se nauči što je to rekurzija te na koji način funkcionira njena implementacija postaje relativno jednostavna te omogućuje kreiranje elegantnih rješenja za probleme koji bi inače zahtjevali kompleksnija rješenja. Korištenje rekurzije u bazi podataka se najčešće pojavljuje kada je potrebno ispisati podatke po određenim razinama. Ispis takvih podataka zahtjeva iteriranje po razinama te ispisivanje svih podataka. Ukoliko programer nije upoznat s rekurzijom ovakav problem može riješiti pisanjem upita na iterativni način, dakle za svaku razinu koja se dohvaća se piše podupit te se na kraju dobiveni rezultati spajaju. Iako na prvi pogled pisanje takvog upita zvuči krajnje nepotrebno oba načina imaju svoje prednosti.

Analiziranjem dobivenih rezultata kod usporedbe brzine izvršavanja rekurzivnih i iterativnih upita dolazimo do zaključka kako je u većini slučajeva rekurzivni upit brži od iterativnog. Kako bi dobili realne rezultate testirali smo upite nad dvije fiktivne organizacije, organizacija „A“ je imala mali broj hijerarhijskih razina a velik broj zaposlenika dok je organizacija „B“ imala veći broj hijerarhija te manji broj zaposlenika. Iako se u većini slučajeva može zaključiti kako je rekurzivni upit brži od iterativnog u organizaciji „A“ postoji trend smanjenja uspješnosti rekurzije. Razlog smanjenju uspješnosti je dohvaćanje velikog broja podataka u čemu je rekurzivni upit sporiji od svog protivnika. S druge strane kod organizacije „B“ primjećujemo trend povećanja uspješnosti rekurzije. Dakle možemo zaključiti kako je korištenje rekurzivnih upita daleko isplativije ukoliko se radi o podacima koji imaju velik broj razina. Na taj način se sprječava pisanje enormnog upita za dohvaćanje podataka iz svih razina te je brzina ispisa podataka također brža. S druge strane ukoliko je veliki skup podataka zapisan na malom broju razina postoji šansa da je pisanje rekurzivnog upita nepotrebna jer bi iterativni način pisanja upita dao bolje rezultate.

Najveća prednost rekurzivnih upita je što upit ne postaje složeniji s povećanjem razina odakle se podaci dohvaćaju. To je upravo iz razloga što upit referencira sam sebe sve dok određeni uvjet nije zadovoljen. U slučaju hijerarhijske strukture i rasporeda materijala uvijet je zadovoljen kada se dođe do zadnje razine, dakle obuhvaćeni su svi podaci po razinama. Korištenje rekurzivnih upita bi uvelike pomoglo velikim organizacijama kod ispisa zaposlenika po organizacijskim razinama. No zbog uočenog trenda smanjenja uspješnosti rekurzivnih upita postavlja se pitanje gdje se brzine izvršavanja rekurzivnih i iterativnih upita izjednačavaju te postoji li mogućnost poboljšanja rekurzivnih upita kako bi uklonili trend smanjenja uspješnosti.

Popis literature

[1] PostgreSQLTutorial (bez dat.) *Learn PostgreSQL Recursive Query By Example.* [Na internetu]. Dostupno: http://www.postgresqltutorial.com/postgresql-recursive-query/ [Pristupano: 30.6.2018].

[2] CS50 Study (bez dat.) *Recursion* [Na internetu]. Dostupno: https://study.cs50.net/recursion [Pristupljeno: 13.8.2018].

[3] D.D. Chamberlin (2012). „Early History of SQL“, IEEE Computer Society [Na internetu] Dostupno: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6359709 [Pristupano: 13.7.2018].

[4] PostgreSQL (bez dat.) *A Brief History of PostgreSQL.* [Na internetu]. Dostupno: https://www.postgresql.org/docs/8.4/static/history.html [Pristupano: 30.6.2018].

[5] T. Baer (2018). „Has the time finally come for PostgreSQL?“ [Na internetu]. Dostupno: https://www.zdnet.com/article/has-the-time-finally-come-for-postgresql/ [Pristupano: 30.6.2018].

[6] Matloob Khushi (2015). “Benchmarking Database Performance for Genomic Data” [Na internetu]. Dostupno: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcb.25049 [Pristupano: 15.7.2018].

[7] M. Horvat, D. Stuparu (2008). „Common Table Expression – WITH Statement“ EDITURA U T PRESS, STR OBSERVATORULUI [Na internetu]. Dostupno: https://www.researchgate.net/publication/242270488_Common_Table_Expression_-_WITH_Statement [Pristupano: 26.6.2018].

[8] V. S. Adamchik (2009). „Binary Trees“ [Na internetu]. Dostupno: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html [Pristupano: 28.8.2018].

[9] Tim NG (2014). „BFS and DFS“ [Na internetu]. Dostupno: https://medium.com/@tim_ng/bfs-and-dfs-52d3cb642a0e [Pristupano: 27.8.2018].

[10] R. Ramakrishnan, J. Gehrke, *Database Management Systems: Third Edition*, McGraw-Hill Irwin, str. 819-826, 2005.

[11] H. Garcia-Molina, J. D. Ullman, J. Widom, *Database Systems: The Complete Book, Second Edition*, Pearson Prentice Hall, str. 437-445, 2009.

[12] Mockaroo (bez dat.) *Random Data Generator* [Na internetu]. Dostupno: https://mockaroo.com/ [Pristupano: 9.9.2018].

[13] Barbkess, „AdventureWorks sample databases“, 2017, [Na internetu]. Dostupno: GitHub, https://github.com/Microsoft/sql-server-samples/releases/tag/adventureworks [Pristupano: 21.7.2018].

[14] K. Wenzel (2016). „Recursive CTEs Explained“ [Na internetu]. Dostupno: https://www.essentialsql.com/recursive-ctes-explained/ [Pristupano: 30.6.2018].

Popis slika

Popis tablica